REMARKS

Claims 9, 13, 18 and 25 have been cancelled without prejudice. Claims 8, 12, 17, 19 and 23 have been amended to more distinctly point out and particularly claim the subject matter of the present invention. Applicant respectfully submits that these amendments should be entered as they do not necessitate a new search because the amendments merely incorporate the subject matter of the cancelled claims into the corresponding independent claims. Thus, claims 1-8, 10-12, 14-17, 19-24 and 26-28 remain pending in the present application. In view of the above amendments and the following remarks, it is respectfully submitted that all of the presently pending claims are allowable.

I. The Rejection Under 35 U.S.C. 103 Should Be Withdrawn

The Examiner rejected claims 1-28 under 35 U.S.C. 103 as being unpatentable over U.S. Patent 5,966,702 to Fresko et al. ("the Fresko reference") in view of U.S. Patent 6,339,841 to Merrick et al. ("the Merrick reference"). *Office Action*, page 2.

The Fresko reference describes a method for pre-processing and packaging class files. Specifically, Fresko describes pre-processing a set of class files into a single file called a multi-class file. *Fresko Reference*, col. 9, lines 10-14. The method is described as taking the set of classes and parsing each of the individual classes to determine the shared set of parameters shared by these classes, *e.g.*, string and numeric constants. *Id.* at col. 9, lines 15-21. These shared values are then stored in a common pool and removed from the pools of the individual classes, thereby reducing the storage requirements for the classes. *Id.* at col. 9, lines 21-25. The multi-class file is then created containing the common pool and all the individual classes having the reduced size. *Id.* at col. 9, lines 36-43. The multi-class file may then be loaded onto the machine using the virtual machine. *Id* at col. 10, lines 16-24. This multi-class loading is described as consolidating the loading of the individual files into a single transaction. *Id.* at col. 10, lines 36-51.

The Merrick reference discloses a system and method for loading classes. Specifically, the Merrick reference describes that a class is comprised of various parts such as a constant pool and methods. *Merrick reference*, col. 3, line 60 - col. 4, line 7. The Merrick

reference teaches that instead of loading all the byte code for a class, some byte code for the methods may be loaded when needed. *Id.* at col. 4, lines 2-4. This is done by loading a method table which includes pointers to the non-loaded methods allowing the non-loaded methods to be loaded when needed. *Id.* at col. 4, lines 39-46.

The Examiner in rejecting the previous arguments raised by the applicant stated that "[a] stack is generally known to store variables as well as status and function call addresses." *Office Action*, page 7. The applicant does not dispute this fact. However, the present invention does not claim that a stack itself is new or unique. The present invention is directed to a novel manner of using a stack to track component loading success and failure. The exemplary embodiments described in the specification focus on the loading of Java classes. However, the use of the stacks in the present invention is not the same as the Java VM runtime stacks as described in the Fresko reference. The Examiner stated that the applicant's argument concerning run-time is moot because the claims do not specify whether or not storing the request is being performed during runtime. *Office Action*, page 7. This argument misses the mark. The applicant was merely pointing out that one of ordinary skill in the art would understand the standard operation of a Java VM runtime stack as described in the Fresko reference. This operation has nothing to do with the storage of information on a stack to track the loading of a class component and the retrieval of that information when a failure occurs. Neither the Fresko reference nor the Merrick reference describe a system where a stack is used to present the user with the name of the module that failed to load, as well as the names of the modules that required the module.

In particular, the Fresko reference and the Merrick reference, either alone or in combination, neither teach nor suggest "a stack to record the [load] request" and "when the loading of the component is unsuccessful, contents of the stack are made available to a user to indicate the unsuccessfully loaded component" as recited in claim 1. Thus, as described in the specification, when a load request is made, the component to be loaded is stored on the stack. If the loading of that component is not successful, the contents of the stack are displayed. These contents will include the name of the class which has not successfully loaded. Again, while the Fresko reference discusses a Java VM runtime stack, it does not disclose the type of information which should be stored in the stack, *i.e.*, the load request, nor that this information should be

displayed when the load is not successful. The Fresko reference does not teach or disclose the functionality described in the present specification and recited in claim 1 to allow a stack to be used to inform a user of a class component which failed to load properly. The Merrick reference does not cure this defect of the Fresko reference.

Furthermore, as the Examiner correctly points out, the Fresko reference also does not teach "wherein when the request has been fulfilled the request is removed from the stack." *Office Action*, page 2. However, the Examiner states that the Merrick reference cures this failing in the Fresko reference. *Id.* at p. 2. In response to the previous arguments made by the applicant, the Examiner states that the function is disclosed when "Merrick in 3:5-9 states, 'One extreme way to discard unused components would be to discard all the components not active on the Java stack.'" *Id.* at p. 7. Initially, the applicant notes that the second reference to "components" in the quoted language should be replaced with "methods." The applicant understands the Merrick reference to suggest the examination of the Java runtime stack and to discard, *i.e.*, remove from the Java runtime heap, any methods that are not currently in use. This is entirely unrelated to the present invention where a load request is removed from the stack when it has been successfully completed. It is respectfully submitted that there is no correlation between discarding a method from a Java runtime heap because it is no longer being used and removing a load request from a stack because the component has been loaded successfully.

Accordingly, for the reasons described above the applicant respectfully submits that neither the Fresko reference nor the Merrick reference, either alone or in combination, teach, disclose or suggest "*a stack to record the request,*" "*when the request has been fulfilled the request is removed from the stack*" and/or "*when the loading of the component is unsuccessful, contents of the stack are made available to a user to indicate the unsuccessfully loaded component*" as recited in claim 1. Thus, the rejection of claim 1 and all claims depending therefrom (claims 2-7) should be withdrawn.

The Examiner also rejected claim 12 for the same reasons as described above for claim 1. *Office Action*, page 2. Claim 12 includes similar limitations as claim 1. Specifically, claim 12 recites "a stack to record a load request for a software component" and a loader which "pops the representation of the software component off of the stack when the load request has

been successfully fulfilled, wherein contents of the stack are made available to a user when fulfillment of the load request has been unsuccessful." As described in the specification, "removing a representation of a class from a stack is referred to as 'popping' a class off the stack." *Specification*, paragraph [0014]. Accordingly, for the reasons described above with reference to claim 1, the applicant respectfully submits that the rejection of claim 12 and all claims depending therefrom (claims 14-16) should be withdrawn.

The Examiner also rejected claim 8 over the Fresko reference in view of the Merrick reference. *Office Action*, page 3-4. Claim 8 includes limitations similar to those recited in claim 1. Specifically, claim 8 recites "placing a representation of the first software module onto a stack," "removing the representation of the first software module from the stack when the first software module has been successfully loaded" and "making contents of the stack available to a user when the loading of one of the first software module and the second software module has been unsuccessful."

Claim 8 also recites "placing, when the first software module is dependent on the second software module, a representation of the second software module onto the stack." Thus, the recitation of claim 8 makes it clear that a representation of both the first and second software module may be entered in the stack. Thus, when the display is shown to the user upon an unsuccessful load attempt, the user will not only see the component that has not successfully loaded, but also those components which depended upon the un-loaded components. Neither the Fresko reference nor the Merrick reference, either alone or in combination, teach or disclose such a system for disclosing dependencies of components which failed to load. Accordingly, for this reason and the reasons described above with reference to claim 1, the applicant respectfully submits that the rejection of claim 8 and all claims depending therefrom (claims 10 and 11) should be withdrawn.

The Examiner also rejected claim 17 over the Fresko reference in view of the Merrick reference. *Office Action*, page 3-4. Claim 17 includes limitations similar to those recited in claim 8. Specifically, claim 17 recites "placing a representation of the first Java class onto a stack," "placing, when the first Java class is dependent on the second Java class, a representation of the second Java class onto the stack," "removing the representation of the first

Java class from the stack when the first Java class has been successfully loaded" and "making contents of the stack available to a user when the loading of one of the first Java class and the second Java class has been unsuccessful." Accordingly, for the reasons described above with reference to claim 8, the applicant respectfully submits that the rejection of claim 17 and all claims depending therefrom (claims 19-22) should be withdrawn.

The Examiner also rejected claim 23 over the Fresko reference in view of the Merrick reference. *Office Action*, page 5-6. Claim 23 includes limitations similar to those recited in claim 1. Specifically, claim 23 recites "a stack to record a load request for a Java class" and a Java class loader which "pops the representation of the Java class off of the stack when the load request has been successfully fulfilled, wherein contents of the stack are made available to a user when fulfillment of the load request has been unsuccessful." Accordingly, for the reasons described above with reference to claim 1, the applicant respectfully submits that the rejection of claim 23 and all claims depending therefrom (claims 24 and 26-28) should be withdrawn.

CONCLUSION

In view of the amendments and remarks submitted above, the applicant respectfully submits that the present case is in condition for allowance. All issues raised by the Examiner have been addressed, and a favorable action on the merits is thus earnestly requested.

Respectfully submitted,

Dated: July 13, 2004

By: _____
Michael J. Marcin (Reg. No. 48,198)

FAY KAPLUN & MARCIN, LLP
150 Broadway, Suite 702
New York, NY 10038
(212) 619-6000 (phone)
(212) 208-6819 (facsimile)